

ABCLib Working Notes No.7

ABCLibScript

利用の手引き ver. 1.00

2004年10月

電気通信大学大学院 情報システム学研究科 情報ネットワーク学専攻
科学技術振興機構 さきがけプログラム 「情報基盤と利用環境」領域

片桐 孝洋

はじめに

ABCLibScript とは、自動チューニング機能付きライブラリを開発するライブラリ作成者の手間を軽減する機能を有する言語（ディレクティブ）です。具体的には、ソースプログラム中に注釈を与えることで、自動チューニングに必要なコードを自動的に生成し、ライブラリ開発者の手間を減らすことを目的にした言語（ディレクティブ）です。

自動チューニング機能付きライブラリ概念は多様な処理に適用可能ですが、当面は並列数値計算処理に特化した言語仕様限定して仕様を作成しております。

1 対象言語

ABCLibScriptは、並列数値計算ライブラリ開発効率向上を目的に設計*された言語（ディレクティブ）です。したがって、数値計算、並列処理に向く言語からのインタフェースを有する必要があります。また、実行可能なプログラムコードを生成することを目的とします。

これらの理由から、数値計算用の言語としては Fortran90、並列処理の言語としては MPI（Fortran インタフェース）が利用できる環境で動作するプログラムコードを生成することを前提とします。

1.2 ライブラリ開発とライブラリ利用の流れ

ABCLibScript を用いたライブラリ開発、および利用者による自動チューニング機能付き数値計算ライブラリ利用の流れは、図 1 のようになります。

* ライブラリ開発に限定せず、任意のユーザプログラムの最適化にも使えます。

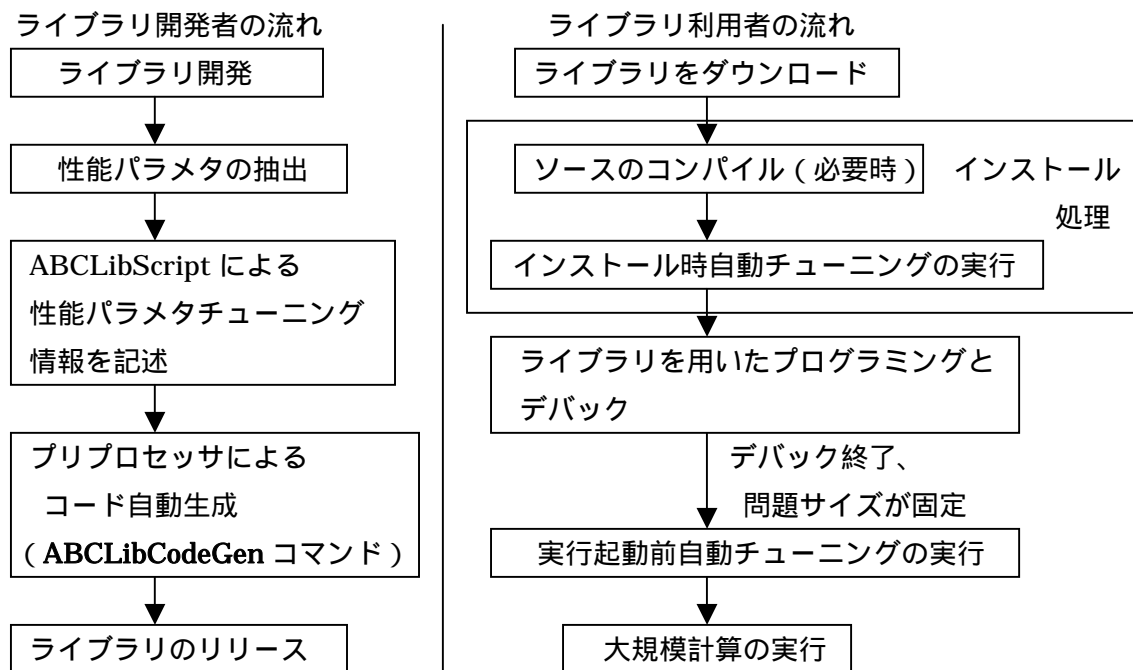


図 1 ABCLibScript を用いたライブラリ開発者および利用者の流れ

1.3 ABCLibScript によるプログラミングの特徴

ABCLibScript は、以下の 4 点に考慮して設計がなされています。

(1) 数値計算処理に特化した自動チューニング指定機能

数値計算処理に特化した自動チューニング機能が用意されており、自動チューニング機能付き数値計算ライブラリの作成が容易となります。

(2) ライブラリ開発者が注釈の形式でプログラム中に指示を与えるチューニング処理指定方式

自動チューニング機能の付加は、ライブラリ開発者が注釈の形式で行います。ゆえに自動チューニング指定が容易であるばかりか、もとのコードのコンパイルと実行を阻害しません。

(3) ライブラリ開発者の注釈を解釈しコードを自動生成するプリプロセッサの提供

ライブラリ開発者の記述した注釈による命令を解釈して自動チューニング処理を付加したコードを自動生成することで、ライブラリ開発者が自動チューニング機構を自動付加されたライブラリのソースコード自体も管理できます。また自動生成されたソースコードを見ることで、付加された処理が理解できます。それゆえ処理の透過性が高く、安心です。

(4) 性能に影響する2種類のパラメタの性能パラメタ (PP) と基本パラメタ (BP) の概念導入

性能パラメタと基本パラメタの概念を導入することにより、自動チューニング処理の見通しがよくなります。また、ライブラリ開発者の意図がシステムに容易に伝わります。

1.4 ABCLibScript のソフトウェアアーキテクチャ

ABCLibScript のユーザインタフェース (Application Programming Interface、API) は、以下の3要素から構成されています。

- 自動チューニング対象を定義する指示子群と自動チューニング処理の詳細定義する補助指示子群
- 自動チューニング情報を定義する環境変数 (システム変数)
- 自動チューニング処理を補助する実行時ルーチン

これらの関係を、図2に示します。

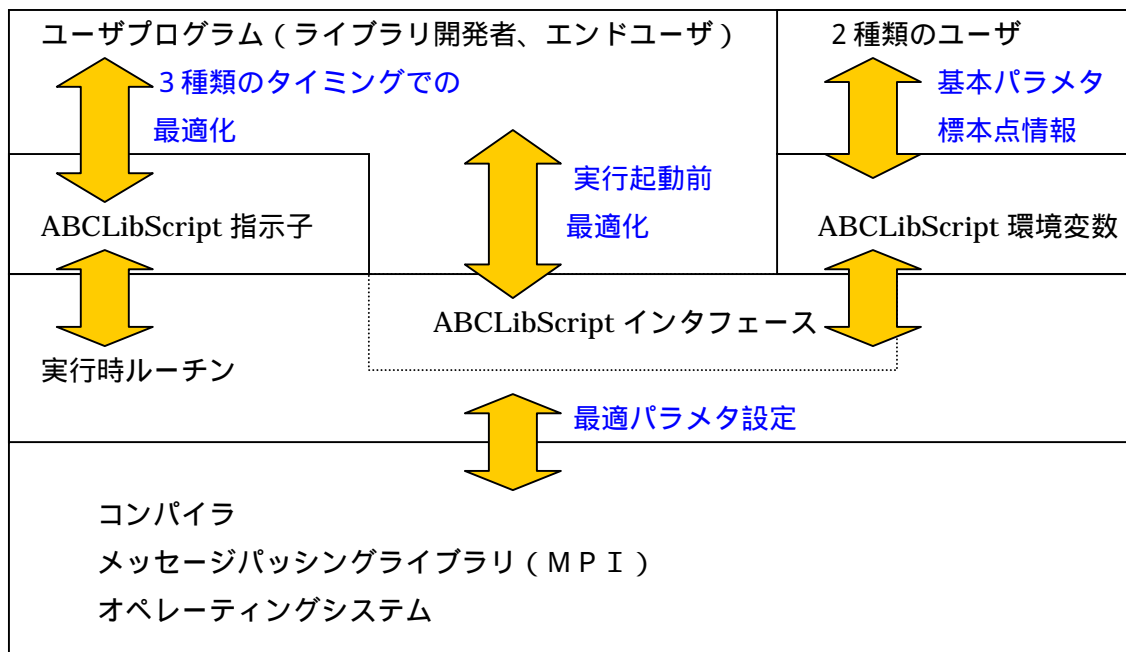


図2 ABCLibScript のソフトウェアアーキテクチャ

2 準備

2.1 システムが想定する自動チューニング機能

ABCLibScript で実現される自動チューニング機能は、以下の 3 機能に分類できます。

- (1) インストール時自動チューニング機能
この機能は、ライブラリインストール時に決定できる性能パラメタの自動チューニングを行う機能です。
- (2) 実行起動前自動チューニング機能
この機能は、エンドユーザにより問題サイズ等の基本パラメタが固定された時点で、性能パラメタについて自動チューニングを行う機能です。
- (3) 実行時自動チューニング機能
この機能は、プログラム中で実際にライブラリが呼ばれた時点で決定できる性能パラメタの自動チューニングを行う機能です。

これらの自動チューニング機能は、サブルーチンとして *ABCLib* プロジェクト (<http://www.abc-lib.org/>) で提案する自動チューニングライブラリの構成方式である FIBER (Framework of Install-time, Before Execute-time and Run-time optimization feature、平成 15 年 1 月特許出願済) ソフトウェアアーキテクチャに従い実装されています。

図 3 に、FIBER の概要を示します。ABCLibScript は、図 3 での自動モデル化ルーチンを除く上記の 3 種類の自動チューニング機能を実現することを目的として開発された言語 (ディレクティブ) です。

また、パラメタ情報の参照には図 4 のような階層があります。すなわち、インストール時自動チューニングルーチンで決定されたパラメタは、実行起動前自動チューニングルーチンと実行時自動チューニングルーチンで参照できます。また、実行起動前自動チューニングルーチンで決定されたパラメタは、実行時自動チューニングルーチンのみでしか参照できません。最後に実行時ルーチンで決定するパラメタは、基本的には実行時ルーチンを除きどのルーチンでも参照はできません[†]。

[†] ただし例外的に、FIBER フィードバックモデルでは、実行時ルーチンで最適化されたパラメタを実行起動前ルーチンで参照して、パラメタを最適化することができます。

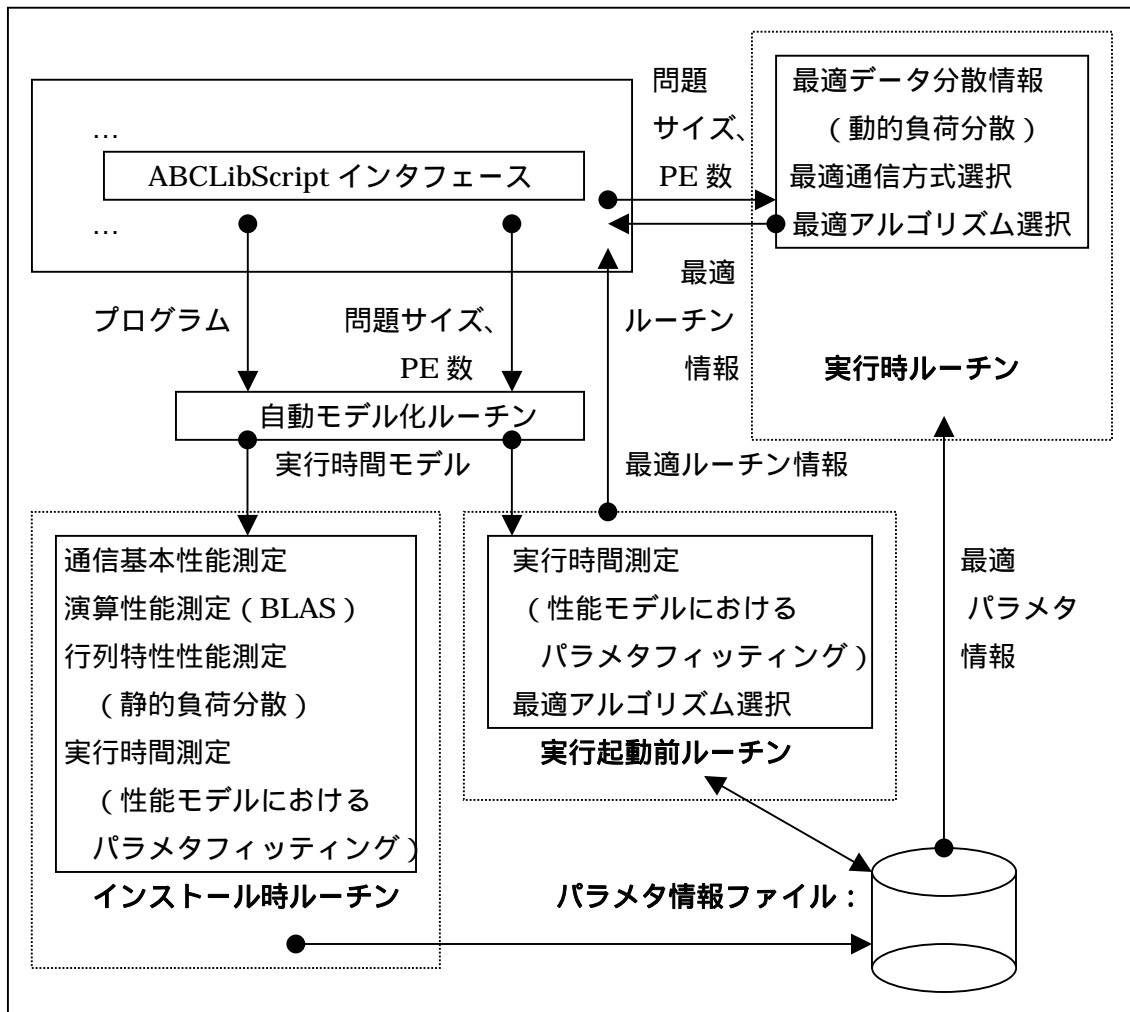


図3 ABCLib プロジェクトにおける自動チューニングライブラリ構成図
(F I B E Rソフトウェアアーキテクチャ)

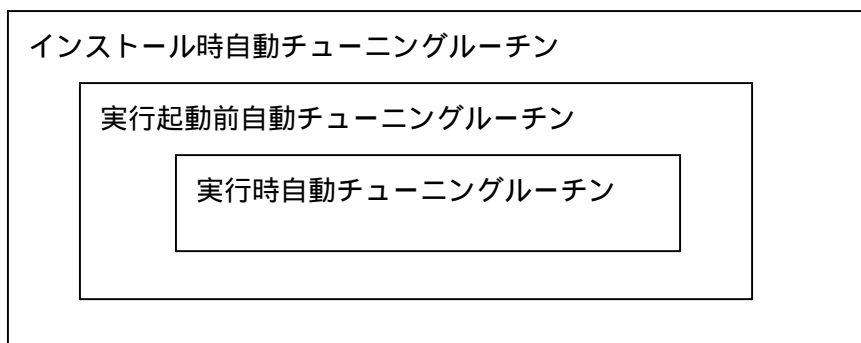


図4 パラメタ情報の参照に関する階層

2.2 ABCLib の自動チューニングの実行順序

ABCLib による自動チューニングは、以下の順番で実行されます。

- (1) インストール時自動チューニングルーチン
- (2) 実行起動前自動チューニングルーチン
- (3) 実行時自動チューニングルーチン

なおこれらの実行順序と異なる場合は、エラーコードを出力して自動チューニングを停止することがあります。

2.3 パラメタの種類

F I B E R では、ユーザ（ライブラリ開発者およびエンドユーザ）が開発するライブラリ、サブルーチン、およびプログラム的一部分について、調整する対象のパラメタの種類が以降に示す 2 種類あります。

- **基本パラメタ (BP)**

エンドユーザがライブラリなどを利用する際に、設定が必須となるパラメタです。例えば、行列サイズやプロセッサ数などです。

- **性能パラメタ (PP)**

エンドユーザがライブラリを利用する際に、設定は必ずしも必要でないが、性能に影響を及ぼすパラメタです。例えば、アンローリング段数などです。

性能情報パラメタは、基本パラメタを固定した場合において、最適な値を見つけられることがユーザにより保証されているパラメタです。

ABCLibScript は、上記の F I B E R における性能パラメタおよび基本パラメタの記述を、ライブラリ開発者が容易に行えるようにするために開発された言語（ディレクティブ）です。したがって、以降で説明する ABCLibScript の表記法は性能パラメタや基本パラメタの宣言・定義・指定をするための言語であるといえます。

2.4 表記法

2.4.1 概略

ABCLibScript では、ソースプログラム中に注釈の形式で処理を記述することで自動チューニング機能を実現します。具体的には、

!ABCLib\$

で始まる注釈行は、ABCLibScript への指示行であるとみなします。

表記法の概略は、以下のとおりです。

```
!ABCLib$ <自動チューニング種類> <機能名> [(対象変数)] region start  
[ !ABCLib$ <機能の詳細> [ sub region start ] ]  
    処理対象のプログラム  
[ !ABCLib$ <機能の詳細> [ sub region end ] ]  
!ABCLib$ <自動チューニング種類> <機能名> [(対象変数)] region end
```

上記の<自動チューニング種類>、<機能名>のことを**指定子**と呼びます。
また上記の<機能の詳細>のことを、**補助指定子**と呼びます。

また、**!ABCLib\$ region start ~ !ABCLib\$ region end**で囲まれた処理のことを、**チューニング領域 (AT 領域)**と呼びます。

2.4.2 指定子

ABCLibScript の指定子の詳細を以下にまとめます。

【指定子一覧】

```
<自動チューニング種類> ::= ( install | static | dynamic | <式> )  
    install : インストール時自動チューニングの指定  
    static : 実行起動前自動チューニングの指定  
    dynamic : 実行時自動チューニングの指定  
    <式> : ::= Fortran90 の文法に従う式であることの指定  
<機能名> ::= ( define | variable | select | unroll )  
    define : パラメタを設定する処理であるという指定  
    variable : 変動するパラメタであるという指定  
    select : 複数の手続きから選択する処理であるという指定  
    unroll : 以下の処理をループアンローリングするという指定
```

2.4.3 補助指定子

指定子 <機能名> の中には、さらなる注釈を記述する必要がある場合があります。それを記述するのが補助指定子ですが、補助指定子 <機能の詳細> には、以下のものがあります。

<機能の詳細> ::= (**name** | **parameter** | **select** | **according** |
varied | **fitting** | **number** | **prepro** | **postpro**)

補助指定子 <機能の詳細> は、指定子によって指定できるか決まります。それを以下に示します。

指定子	指定可能な補助指定子
define	name、parameter、number、prepro、postpro
variable	name、parameter、varied、fitting、number、prepro、postpro
select	name、parameter、select、according、number、prepro、postpro
unroll	name、parameter、varied、fitting、number、prepro、postpro

各補助指定子の詳細は、以下のとおりです。

【補助指定子一覧】

name <文字列> : チューニング領域の名称を記述 (全ての機能で利用可能)

parameter (<属性指定> <変数名>、[<属性指定> <変数名>、...])

: 性能特性ファイルに出力か、性能特性ファイルから入力するパラメタを指定

<属性指定> : := [**in** | **out** | **bp**]

in : 入力される (外部で定義され参照される) パラメタ

out : 出力する (このチューニング領域で定義される) パラメタ

bp : 基本パラメタ

(全ての機能で利用可能)

select sub region (**start** | **end**) : 選択する手続きであることを指定

(機能名 **select** 指定時)

according (<条件式> | **estimated**) : 以降に指定する基準で、手続きを選択することを指定

<条件式> : := [(**min**(<変数名>) | **condition**(<条件>)) <接続演算子>]

<接続演算子> : := [**.and.** | **.or.**] <条件式>

estimated <数式> : ユーザが定義した選択に関するコスト (数式) を基に、最適な処理を選択する処理であることを指定

(機能名 **select** 指定時)

varied (<変数> [, <変数>]) **from** X **to** Y

指定するパラメタの変動範囲 (X から Y まで) の指定

(機能名 **variable**、 **unroll** 指定時)

fitting <方式> **sampled** <範囲> : パラメタの推定に利用する方式を指定

<方式> : := [**least-squares** <次数> | **user-defined** <数式> | **auto**]

least-squares : 多項式による最小二乗法でパラメタ推定することを指定します。 <次数> で、多項式の次数を設定します。

user-defined : ユーザ指定による数式を用いて、最小二乗法で推定します。

auto : システム側にパラメタの推定を任せます。

<範囲> : := [<数値> | **auto**] パラメタ推定に必要な範囲を指定します。

なお <方式> = **auto** 指定時は省略できます。

<数値> : 具体的なパラメタの数値を指定します

auto : パラメタのサンプリング間隔を自動決定します。

fitting 補助指定子省略時は、**varied** 補助指定子で指定した範囲全てを測定 (= 全探索法) して最適パラメタを決めます。

(機能名 **variable**、 **unroll** 指定時)

(次頁に続く)

number <番号> : 当該チューニング領域に対するチューニング順番を、<番号>で指定する。省略時は出現順となる。入れ子指定子の場合、最外側の指定子しか記述できない (全ての機能で利用可能)

prepro sub region (start | end) : チューニング領域を呼び出す前に適用する処理を指定 (全ての機能で利用可能)

postpro sub region (start | end) : チューニング領域を呼び出した後に適用する処理を指定 (全ての機能で利用可能)

debug (<変数>、[<変数>、...]) : チューニング領域が実行されるときに、デバック表示する変数を指定 (全ての機能で利用可能)

<変数> : := [**bp** | **pp** | **任意変数**]

bp : 基本パラメタ情報を表示する

pp : 性能パラメタ情報を表示する

任意変数 : 記述された変数情報を表示する

【プログラム例 1】 行列積の 1 段から 16 段のアンローリングをインストール時ルーチンで自動チューニングする。パラメタ推定には 5 次多項式を利用して最小二乗法で決定する。また標本点は、1-5、8、16 を指定する。デバック表示する対象は、性能パラメタ (アンローリング段数) 情報を指定する。

```

!ABCLib$ install unroll region start
!ABCLib$ name MyMatMul
!ABCLib$ varied (i,j) from 1 to 16
!ABCLib$ fitting least-squares 5 sampled (1-5, 8, 16)
!ABCLib$ debug (pp)
do i=1, n
  do j=1, n
    do k=1,n
      A(i,j) = A(i,j) + B(i,k) * C(k,j)
    enddo
  enddo
enddo
!ABCLib$ install unroll (i,j) region end

```

3 ABCLibScript による自動チューニングライブラリの実装

3.1 初期化インタフェースの説明

ABCLibScript では、自動チューニングを実行するためのインタフェースを用意しており、それを利用することでユーザは自動チューニングの指定を行います。

具体的には、自動チューニングをするために ABCLibScript が提供する、以下のインタフェースを利用します。

- **ABCLib_ATexec** (ABCLib_ATkinds, ABCLib_ATroutines)

ABCLib_ATexec 手続きは、ABCLib_ATkinds で指定した自動チューニングを、ABCLib_ATroutines で指定したチューニング領域について実行する手続きです。

引数 ABCLib_ATkinds は、自動チューニングの種類を指定するための引数です。ヘッダファイル ABCLibScript.h で定義された以下の3種の定数が指定できます。

ABCLib_INSTALL : インストール時自動チューニング
ABCLib_STATIC : 実行起動前時自動チューニング
ABCLib_DYNAMIC : 実行時自動チューニング
ABCLib_ALL : すべての自動チューニング

引数 ABCLib_ATroutines は、どのチューニング領域の処理についてか指定する引数です。この引数はヘッダファイル ABCLibScript.h で定義されている型 ABCLib_ATname を用いてユーザが宣言するか、ABCLibScript.h で common 定義されている大域変数

ABCLib_AllRoutines : すべてのルーチン用
ABCLib_InstllRoutines : インストール時自動チューニングルーチン用
ABCLib_StaticRoutines : 実行起動前時自動チューニングルーチン用
ABCLib_DynamicRoutines : 実行時自動チューニングルーチン用

を利用して指定します。

自動チューニング処理部情報の代入は、サブルーチン ABCLib_ATset で行います。

なお、インストール時自動チューニング、実行起動前自動チューニングでは、この ABCLib_ATexec が呼ばれた時に自動チューニングが行われます。ところが実行時自動チュ

ーニングについては、実行設定のみ行われます。実際の自動チューニングは、当該部分が呼ばれた時点で行われます。

【使用例】

```
!ABCLib$ call ABCLib_ATexec (ABCLib_INSTALL, ABCLib_InstallRoutines)
```

：インストール時最適化を行います。

● **ABCLib_ATset** (ABCLib_ATkinds, ABCLib_ATroutines)

ABCLib_ATset 手続きは、ABCLib_ATkind で指定した、自動チューニング実行指定したチューニング領域名を、ABCLib_ATroutines に設定する手続きです。

引数 ABCLib_ATkinds は、自動チューニングの種類を指定するための引数です。ヘッダファイル ABCLibScript.h で定義された、以下の3種類の定数が指定できます。

ABCLib_INSTALL : インストール時自動チューニング

ABCLib_STATIC : 実行起動前時自動チューニング

ABCLib_DYNAMIC : 実行時自動チューニング

ABCLib_ALL : すべての自動チューニング

引数 ABCLib_ATroutines に、対象となるチューニング領域名が代入されます。

なお、特定のルーチン名を除外したい場合には、以下のサブルーチン ABCLib_ATdel を用います。

● **ABCLib_ATdel** (ABCLib_ATroutines, DelName)

ABCLib_ATdel 手続きは、ABCLib_ATroutines で指定したチューニング領域名情報が入っている変数から、DelName で指定するチューニング領域名を削除する手続きです。

引数 DelName に削除したいチューニング領域名を記述します。なおチューニング領域名は、注釈の形式で各チューニング領域に記述しておきます(後述)。

【使用例】

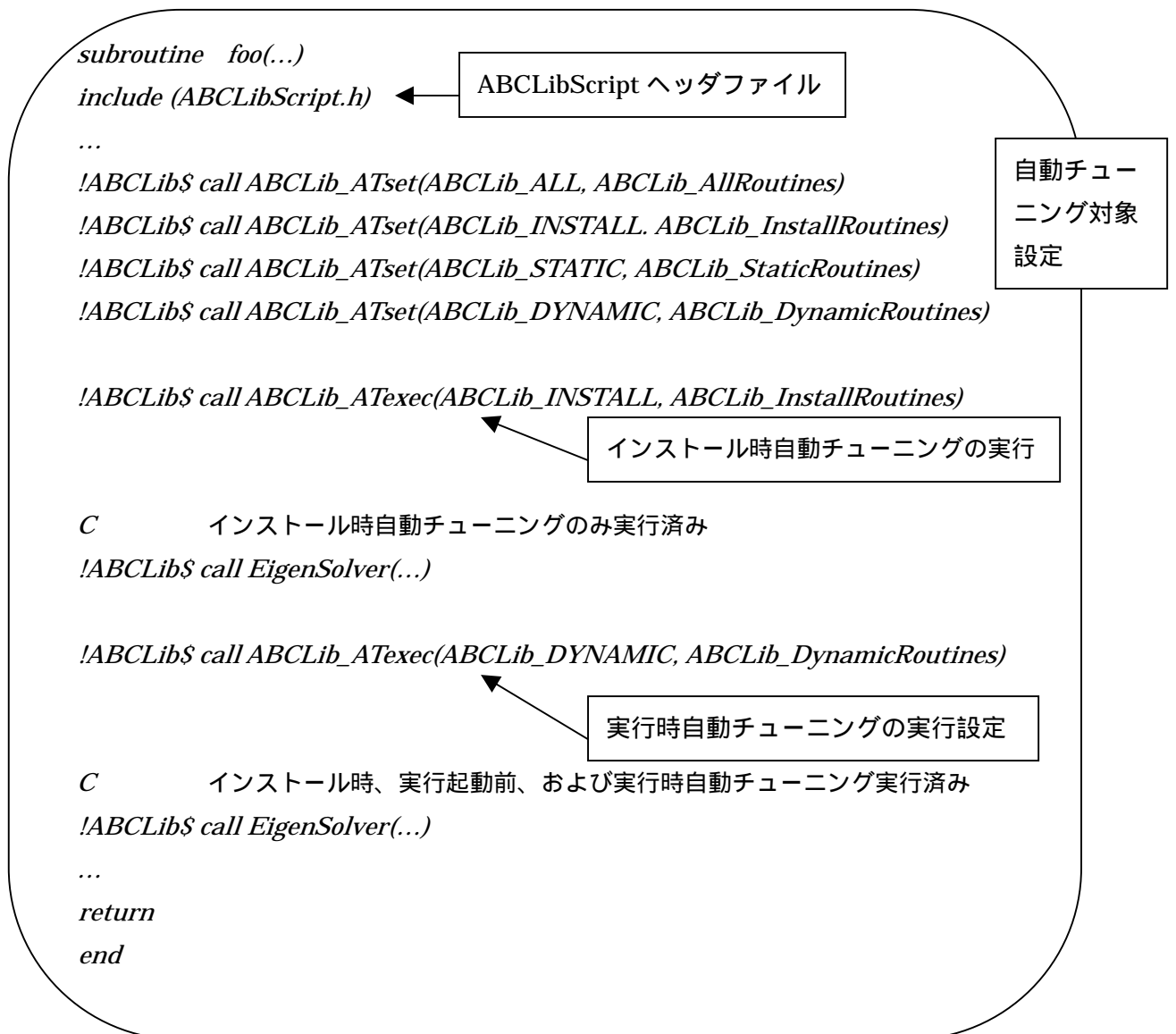
```
!ABCLib$ call ABCLib_ATdel(ABCLib_InstallRoutines, "MyMatMul")
```

: インストール時最適化を行う候補から、MyMatMul と名付けたチューニング領域を除外します。

3.2 実装方法の概略

ここでは、具体的に例をあげて各機能の説明をします。

ライブラリ開発者が所有するサブルーチン EigenSolver は、自動チューニングを行うライブラリ開発者サブルーチン foo 中で ABCLibScript のインタフェースを用いて、以下のよ
うな流れで記述できます。



この一方でエンドユーザは、ライブラリ開発者が提供したライブラリの機能である EigenSolver 利用に関して、実行起動前自動チューニングを行うライブラリ利用者のサブルーチン `pooh` 中で、`ABCLibScript` のインタフェースを用いて、以下のような流れで記述できます。

```
subroutine pooh(...)
include (ABCLibScript.h) ← ABCLibScript ヘッダファイル
...

C ライブラリ開発者が指定する基本パラメタの固定
  N_TUNESIZE_START = 1234
  N_TUNESIZE_END = 1234
  call ABCLib_ATexec(ABCLib_STATIC, ABCLib_StaticRoutines) ← 実行起動前自動チューニングの実行

C インストール時および実行起動前自動チューニング実行済み
  call EigenSolver(...)
...
return
end
```

3.2.1 インストール時自動チューニングルーチンの利用例

インストール時自動チューニングルーチンは、基本的にはライブラリインストール終了後に1度実行されることを想定しています。したがってデフォルトでは、2度以上実行を指定しても処理はなされません。もし処理をしたい場合は、以下の処理の初期化インタフェース `ABCLib_ATInstallInit` を呼び初期化します。

- **ABCLib_ATInstallInit** (`ABCLib_InstallRoutines`)

`ABCLib_ATInstallInit` 手続きは、`ABCLib_InstallRoutines` で指定されるインストール時自動チューニング領域名のチューニングを未実行とします。

【使用例】

```
!ABCLib$ call ABCLib_ATInstallInit(ABCLib_InstallRoutines)
```

インストール時自動チューニングの実行を、まだ行っていないことにします。

インストール時自動チューニングルーチンでは、【プログラム例 1】で示したインストール時に行うパラメタ推定処理のほかに、インストール時に測定できるパラメタを測定する処理を想定しています。この指定は、以下の define 機能で記述できます。

【プログラム例 2】

インストール時ルーチンにおける define 機能を用いたパラメタ決定処理

```
!ABCLib$ install define (CacheSize, CacheLine) region start  
!ABCLib$ name SetCachePram  
!ABCLib$ parameter (out CacheSize, out CacheLine)  
...  
CacheSize =....  
CacheLine=....  
!ABCLib$ install define (CacheSize, CacheLine) region end
```

インストール時ルーチンでは、これら指定されたパラメタの値を**パラメタ情報ファイル** (Parameter Information File) に保存します。ファイル名は `ABCLib_InstallPram.dat` で、テキスト形式で保存されています。

例：パラメタ情報ファイル `ABCLib_InstallParam.dat` の中身

```
(SetCacheParam  
  (CacheSize 64)  
  (CacheLine 8)  
)
```

3.2.2 実行起動前自動チューニングルーチンの利用例

実行起動前自動チューニングでは、エンドユーザがライブラリ実行前に与える詳細な情報をもとにパラメタを決定します。また実行起動前自動チューニングは、2.4 節で説明した基本パラメタの値に関して、エンドユーザが保証した値に固定した条件でのパラメタ調整機能であるといえます。

ユーザがライブラリ実行起動前に与えるパラメタは、パラメタ情報ファイル ABCLib_StaticParamDef.dat 中で指定します。基本パラメタについては、プログラム中から代入文で設定できます。ここで**基本パラメタ**とは、インストール時および実行起動前自動チューニングで必須となるパラメタのことで、デフォルトではシステムで指定した変数名を用いて内部参照されます。

以下に、デフォルト設定している基本情報パラメタを示します。

【デフォルト基本パラメタ一覧】

<p><デフォルト基本パラメタ> ::= (ABCLib_NUMPROCS ABCLib_STARTTUNESIZE ABCLib_ENDTUNESIZE ABCLib_SAMPDIST)</p> <p>ABCLib_NUMPROCS : 整数型。利用するプロセッサ台数の情報。</p> <p>ABCLib_STARTTUNESIZE : 整数型。デフォルト基本パラメタに対する、 自動チューニングを始める問題サイズ</p> <p>ABCLib_ENDTUNESIZE : 整数型。デフォルト基本パラメタに対する、 自動チューニングを終える問題サイズ、</p> <p>ABCLib_SAMPDIST : 整数型。デフォルト基本パラメタに対する、 自動チューニング時に増加させる問題サイズ (増分) の大きさ</p>

なお、基本パラメタを設定しないと、実行起動前自動チューニングは実行できません。また、インストール時自動チューニングは ABCLib_NUMPROCS、ABCLib_STARTTUNESIZE、ABCLib_ENDTUNESIZE、ABCLib_SAMPDIST、を指定しないと実行できません。

ライブラリ開発者が基本パラメタを新たに設定したい場合は、以下の ABCLib_BPset、および ABCLib_BPsetName 手続きを利用します。

● ABCLib_BPset(BPvalName)

ABCLib_BPset 手続きは、引数 BPvalName に指定した基本パラメタ名を、新たな基本パラメタとします。

【使用例】

```
!ABCLib$ call ABCLib_BPset ("nprocs")
```

：変数 nprocs を基本パラメタとします。

● ABCLib_BPsetName(Kind, BPvalName, Name)

ABCLib_BPsetName 手続きは、引数 BPvalName に指定した基本パラメタ名について引数 Kind で指定される種類の基本パラメタに関する情報変数を、新たに引数 Name で指定される名前とします。

ここで、引数 Kind は、以下のとおりです。

Kind ::= [**STARTTUNESIZE** | **ENDTUNESIZE** | **SAMPDIST**]

STARTTUNESIZE : 基本パラメタ BPvalName に関する標本開始点情報

ENDTUNESIZE : 基本パラメタ BPvalName に関する標本終了点情報

SAMPDIST : 基本パラメタ BPvalName に関する標本点間隔情報

【使用例】

```
!ABCLib$ call ABCLib_BPsetName("STARTTUNESIZE", "nprocs",
```

```
!ABCLib$ & "ABCLib_NprocsStartSize")
```

：基本パラメタ nprocs における自動チューニングに関する標本開始点情報変数名を ABCLib_NprocsStartSize とします。

● ABCLib_BPsetCDF(BPvalName, CFDFKind)

ABCLib_BPsetCDF 手続きは、引数 BPvalName に指定した基本パラメタ名に関する標本点以外の推定方式を、引数 CFDFKind で指定される種類のコスト定義関数に設定します。ここで、引数 CFDFKind は、以下のとおりです。

CFDFKind ::= [**least-squares** <次数> | **user-defined** <数式> | **auto**]

least-squares <次数> : 多項式による最小二乗法でパラメタ推定することを指定します。<次数> で多項式の次数を指定します。

user-defined <数式> : ユーザによる数式を用いて最小二乗法で推定します。

auto : システム側にパラメタ推定を任せます。

なおデフォルトでは、当該チューニング領域で指定されたコスト定義関数を、そのまま

利用します。

【使用例】

```
!ABCLib$ call ABCLib_BPsetCFD("nprocs", "least-squares 5")
```

: 基本パラメタ nprocs に対するパラメタ推定方式を、5 次多項式として最小二乗法で推定します。

【プログラム例 3】 4 台のプロセッサを用いて、1024、 2048、 3072 次元のパラメタを実行起動前チューニングしたい場合の、基本パラメタである標本点情報の指定

```
!ABCLib$ ABCLib_TUNESTATIC = 1  
!ABCLib$ ABCLib_NUMPROCS = 4  
!ABCLib$ ABCLib_STARTTUNESIZE = 1024  
!ABCLib$ ABCLib_ENDTUNESIZE = 3072  
!ABCLib$ ABCLib_SAMPDIST = 1024  
!ABCLib$ call ABCLib_ATexec(ABCLib_STATIC, ABCLib_StaticRoutines)
```

もうひとつの指定方法として、ABCLib_StaticParamDef.dat ファイル中に、

```
(BasicParam  
(ABCLib_TUNESTATIC 1)  
(ABCLib_NUMPROCS 4)  
(ABCLib_STARTTUNESIZE 1024)  
(ABCLib_ENDTUNESIZE 3072)  
(ABCLib_SAMPDIST 1024)  
)
```

と記述する方法もあります。

実行起動前ルーチンでは、unroll 機能の利用が想定されています。その利用例を以下に示します。

【プログラム例 4 a】

実行起動前ルーチンにおいて、以下のプログラムを 16 段までアンローリングして最適なパラメタを決定する。パラメタ測定は、1 - 16 段全て (= 全探索法) とする。ただし、基本パラメタの標本点に関する初期値は、【プログラム例 3】のように指定済みとする。ループ導入変数が変数 n しかないので、変数 n がデフォルト基本パラメタであると解釈する。

```
!ABCLib$ static unroll (i,j) region start  
!ABCLib$ name MyMatMul  
!ABCLib$ varied (i,j) from 1 to 16  
do i=1, n  
  do j=1, n  
    do k=1, n  
       $A(i,j) = A(i,j) + B(i,k)*C(k,j)$   
    enddo  
  enddo  
enddo  
!ABCLib$ static unroll (i,j) region end
```

この例の場合、出力パラメタは最適な (= 最もこの処理が高速となる) i, j の値となります。

実行起動前自動チューニングルーチンでは、この最適化されたパラメタを、パラメタ情報ファイルである ABCLib_StaticParam.dat に出力します。

例：パラメタ情報ファイル ABCLib_StaticParam.dat の中身:

```
(MyMatMul  
  (ABCLib_NUMPROCS  4)  
  (ABCLib_SAMPDIST  1024)  
  (ABCLib_PROBSIZE  1024  
    (MyMatMul_I  4)  
    (MyMatMul_J  8) )  
  (ABCLib_PROBSIZE  2048  
    (MyMatMul_I  4)
```

```

(MyMatMul_J 9) )
(ABCLib_PROBSIZE 3072
(MyMatMul_I 5)
(MyMatMul_J 10) )
)

```

実行起動前自動チューニングルーチンのパラメタ決定には、プログラム中で記述指定しているパラメタのみを利用して決定することもできますし、インストール時自動チューニングルーチンのパラメタ（パラメタ情報ファイル上のデータ）を呼び出し、それをもとにして決めることもできます。【プログラム例5】に、その例を示します。

【プログラム例4b】

実行起動前ルーチンにおいて、以下のプログラムを16段までアンローリングし最適なパラメタを決定する。パラメタ測定は、1 - 16段全て（=全探索法）とする。ただし基本情報パラメタは、【プログラム例3】のように指定済みとする。

以下の例では、ループ終了変数が n だけではない（変数 `nprocs` も存在する）ため、変数 n と変数 `nprocs` のどちらがデフォルト基本パラメタであるか分からない。したがって、`parameter` 補助指定子により、ユーザによる基本パラメタの指定が必要になる。

```

!ABCLib$ static unroll (i,j) region start
!ABCLib$ name MyMatMul
!ABCLib$ parameter(bp n)
!ABCLib$ varied (i,j) from 1 to 16
do i=1, n/nprocs
  do j=1, n
    do k=1, n
      A(i,j) = A(i,j) + B(i,k)*C(k,j)
    enddo
  enddo
enddo
!ABCLib$ static unroll (i,j) region end

```

【プログラム例 4c】

実行起動前ルーチンにおいて、以下のプログラムを 16 段までアンローリングして最適なパラメタを決定する。パラメタ測定は、1 - 16 段全て (= 全探索法) とする。

ただし基本情報パラメタは、【プログラム例 3】のように指定済みとする。

以下の例では、基本パラメタは変数 n だけでなく、変数 $nprocs$ も基本パラメタであると指定する。

```
!ABCLib$ call ABCLib_BPsetVal("nprocs")
!ABCLib$ call ABCLib_BPsetName(STARTTUNESIZE, "nprocs",
!ABCLib$ &          "ABCLib_NprocsStartSize")
!ABCLib$ call ABCLib_BPsetName(ENDTUNESIZE, "nprocs",
!ABCLib$ &          "ABCLib_NprocsEndSize")
!ABCLib$ call ABCLib_BPsetName(SAMPDIST, "nprocs",
!ABCLib$ &          "ABCLib_NprocsSampDist")
!ABCLib$ ABCLib_NprocsStartSize = 1
!ABCLib$ ABCLib_NprocsEndSize = 8
!ABCLib$ ABCLib_NprocsSampDist = 1
!ABCLib$ static unroll (i,j) region start
!ABCLib$ name MyMatMu
!ABCLib$ parameter(bp n, bp nprocs)
!ABCLib$ varied (i,j) from 1 to 16
do i=1, n/nprocs
  do j=1, n
    do k=1, n
       $A(i, j) = A(i, j) + B(i, k) * C(k, j)$ 
    enddo
  enddo
enddo
!ABCLib$ static unroll (i,j) region end
```

【プログラム例 5】

インストール時自動チューニングルーチンのパラメタCacheSizeを参照し、さらに実行起動前にエンドユーザが固定する問題サイズ、プロセッサ台数、の基本パラメタ情報を利用して、最適な実装方式を決定する。このとき、ある基準（この場合はユーザが与える実行時間予測式）で自動選択[‡]する。

```
!ABCLib$ static select region start
!ABCLib$ name ATfromCacheSize
!ABCLib$ parameter (in CacheSize, in ABCLib_PROBSIZE,
!ABCLib$ & in ABCLib_NUMPROC)
!ABCLib$ select sub region start
!ABCLib$ according estimated
!ABCLib$ & 2.0d0*CacheSize*ABCLib_PROBSIZE*ABCLib_PROBSIZE
!ABCLib$ & / (3.0d0*ABCLib_NUMPROC)
対象処理 1
!ABCLib$ select sub region end
!ABCLib$ select sub region start
!ABCLib$ according estimated 4.0d0*CacheSize*ABCLib_PROBSIZE
!ABCLib$ & *dlog(ABCLib_PROBSIZE) / (2.0d0*ABCLib_NUMPROC)
対象処理 2
!ABCLib$ select sub region end
!ABCLib$ static select region end
```

3.2.3 実行時自動チューニングルーチンの利用例

実行時自動チューニングルーチンは、ライブラリ実行時に得られる情報をもとにして、性能パラメタの決定を行うルーチンです。この性能パラメタ決定時には、プログラム中で指定したパラメタ、インストール時自動チューニングルーチンで決定したパラメタ、実行起動前自動チューニングルーチンで指定したパラメタ、の参照が可能です。

実行時ルーチンにおいて、利用が想定される select 機能を例で説明します。

[‡] この場合は、実行時間に関する基準で選択しています。一般的に、実行時間のみが A T 領域の選択基準でない点に注意してください。例えば、サブルーチンに関する値段（価格）も選択基準となり得ます。このような選択基準も、select 指定子で記述できます。

【プログラム例 6】

プログラム中で定義するパラメタ *eps*、*iter* をもとに、最適な処理を選択する。
具体的には、*eps* が最小となるような処理を条件 *iter*<5 以内に決定する。

```
!ABCLib$ dynamic select (eps, iter) region start  
!ABCLib$ name PricondSelect  
!ABCLib$ parameter (in eps, in iter)  
!ABCLib$ according min (eps) .and. condition (iter < 5)  
!ABCLib$ select sub region start  
 対象処理 1  
  eps=...  
!ABCLib$ select sub region end  
!ABCLib$ select sub region start  
 対象処理 2  
  eps=...  
!ABCLib$ select sub region end  
!ABCLib$ dynamic select (eps, iter) region end
```

● **ABCLib_DynPefThis(Name)**

ABCLib_DynPefThis 手続きは、引数 Name に指定した実行時自動チューニングを行うチューニング領域名の最適化について、この手続きが記述された場所で行うことを指定します。なお本手続き利用時においては、チューニング領域 Name の実行時において、最適化済みパラメタを用いて実行がなされます。すなわち、Name で指定された当該チューニング領域ではパラメタチューニングはなされません。

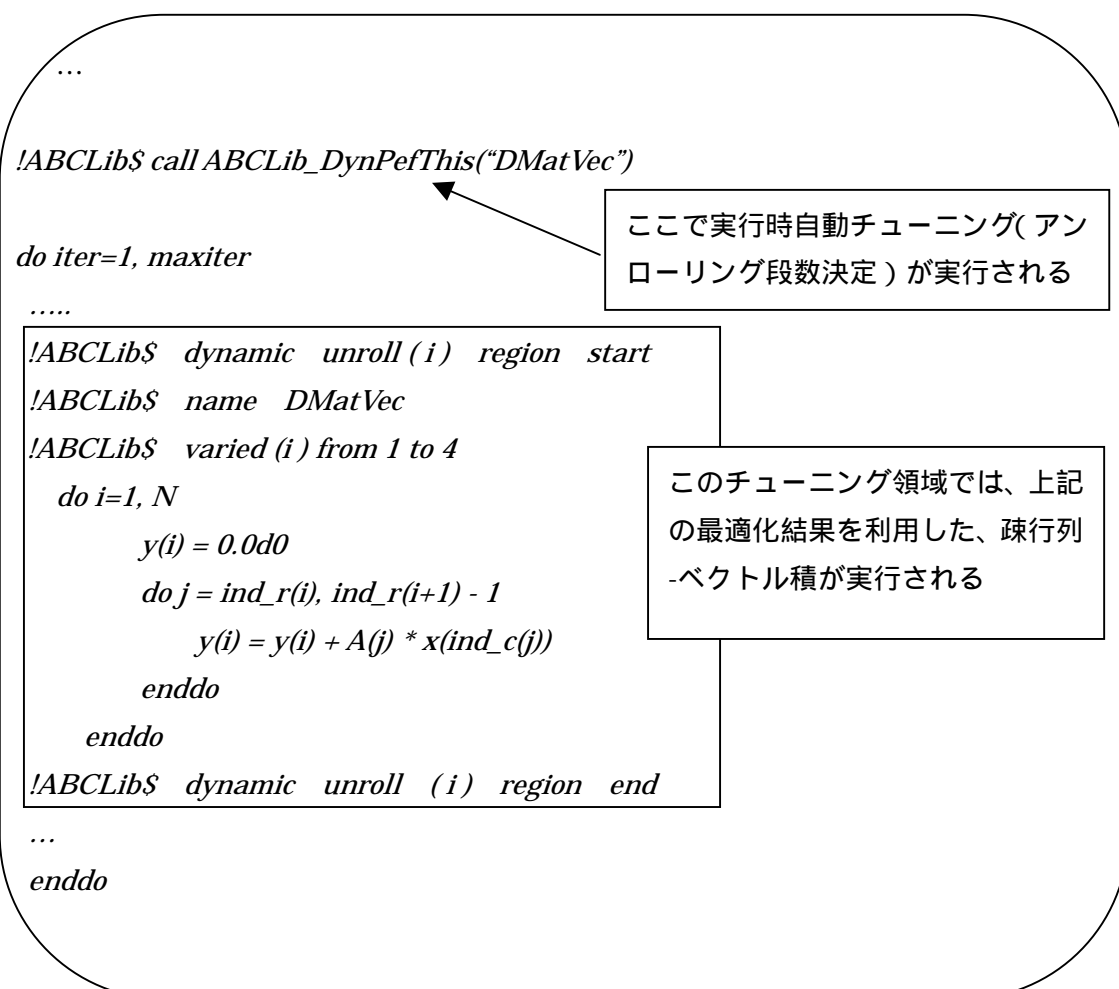
【使用例】

```
!ABCLib$ call ABCLib_DynPefThis ( "DMatVec" )
```

: 自動チューニング領域 DMatVec の自動チューニングを、この箇所で行う。

【プログラム例 7】

最適なループアンローリング段数決定に関して、以前に行った実行時最適化の最適化情報を再利用し、当該 A T 領域で最適化された処理を実行する。



3.3 自動チューニングコード生成コマンド利用例

実際の自動チューニング機能付き並列数値計算ライブラリのコード (Fortran90 + MPI) を生成するには、以下のようにします。

```
>ABCLibCodeGen test.f
```

ここで test.f は、 ABCLibScript のディレクティブを含むプログラムとします。

上記のコマンド実行後、カレントディレクトリの下に ABCLib ディレクトリが作られ、そのディレクトリの下に、以下のファイルが作成されます。

<pre>./ABCLib/ABCLib_test.f : test.f に自動チューニングコードを埋め込んだ Fortran90+MPI ソースコード ./ABCLib/ABCLib_InstallRoutines.f : test.f から抽出した、インストール時 自動チューニングを行うサブルーチン群 ./ABCLib/ABCLib_StaticRoutines.f : test.f から抽出した、実行起動前自動 チューニングを行うサブルーチン群 ./ABCLib/ABCLib_DynamicRoutines.f : test.f から抽出した、実行時自動 チューニング処理を行うサブルーチン群 ./ABCLib/ABCLib_ControlRoutines.f : 自動チューニングコード制御サブルーチン群</pre>
--

プリプロセッサ起動時にすでにこれらのファイルがある場合、重複していないソースコードとサブルーチンは、上記のファイルに追加して記録されます。

3.3.1 実行時オプション

ABCLibCodeGen は、以下に示す実行時オプションを指定できる。

<pre>- debug OFF : デバック用のコードを生成しない (デフォルト値) ON : ABCLib_PRINT 変数で指定されるデバックレベルxの デバック用コードを生成する。 - visualization OFF : 自動チューニング軌跡ファイルを出力しない (デフォルト値) ON : 自動チューニング軌跡ファイルを出力し、そのファイルを利用して ビジュアライゼーションする</pre>

- visualization ON で作成される自動チューニング軌跡ファイル名は

- ABCLibATlog.dat

です。

【使用例】

```
> ABCLibCodeGen - debug ON - visualization ON test.f
```

ABCLibScript による指示子が記入されているプログラム test.f に対して、デバック用コードを生成し、かつ自動チューニングモードでの軌跡を自動チューニング軌跡ファイルに残すことを指定する。この自動チューニング軌跡ファイルを利用して、ビジュアライザ (*VizABCLib*) を用いて動的なチューニング経過が閲覧できる。

4 ABCLibScript の内部仕様

4.1 システム変数

ABCLibScript がシステムとして所有している変数の一覧を以下にのせます。なお、これらの変数は予約語であるのでユーザは定義できません。

【システム変数一覧 (予約語)】

[チューニング種類指定]

ABCLib_ALL : 整数型、実体は 0。全チューニング (インストール時、実行起動前、実行時) をさす
ABCLib_INSTALL : 整数型、実体は 1。インストール時最適化をさす。
ABCLib_STATIC : 整数型、実体は 2。実行起動前最適化をさす。
ABCLib_DYNAMIC : 整数型、実体は 3。実行時最適化をさす。

[チューニング領域名保存]

ABCLib_AllRoutines : 文字列型。全チューニング領域名が入る。
ABCLib_InstallRoutines : 文字列型。インストール時自動チューニングをするチューニング領域名が入る。
ABCLib_StaticRoutines : 文字列型。実行起動前自動チューニングをするチューニング領域名が入る。
ABCLib_DynamicRoutines : 文字列型。実行時自動チューニングをするチューニング領域名が入る。

[デフォルト基本パラメタの標本点]

ABCLib_NUMPROCS : 整数型。利用するプロセッサ台数が入る。

ABCLib_STARTTUNESIZE : 整数型。デフォルト基本パラメタにおける標本点
開始値が入る。

ABCLib_ENDTUNESIZE : 整数型。デフォルト基本パラメタにおける標本点
終了値が入る。

ABCLib_SAMPDIST : 整数型。デフォルト基本パラメタにおける標本点間隔
(増分)が入る。

[システム制御]

ABCLib_TUNESTATIC : 論理型。実行起動前自動チューニングをするかどうかを指定
する。 **.true.** : 実行する。 **.false.** : 実行しない

ABCLib_TUNEDYNAMIC : 論理型。実行時自動チューニングをするかどうかを指定
する。 **.true.** : 実行する。 **.false.** : 実行しない

ABCLib_DEBUG : 整数値。デバック用プリントのレベルを指定。

0 : なし。 **1** 以上の値 **x** : レベル **x** のデバック用プリントをする。

4.2 入出力ファイル

ここでは ABCLibScript が扱う入出力ファイル (パラメタ情報ファイル) について説明します。入出力ファイルには、ABCLibScript が自動生成するファイル (システム指定ファイル) と、デバックなどの理由でユーザが指定するファイル (ユーザ指定ファイル) の2種類があります。

システム指定ファイル :

ABCLib_InstallParamX.dat : インストール時自動チューニングルーチン
のパラメタ出力用

ABCLib_StaticParamX.dat : 実行起動前自動チューニングルーチン
のパラメタ出力用

ユーザ指定ファイル：

ABCLib_InstallParamDefX.dat : インストール時自動チューニングルーチンのパラメタ指定用

ABCLib_StaticParamDefX.dat : 実行起動前自動チューニングルーチンのパラメタ指定用

ABCLib_DynamicParamDefX.dat : 実行時自動チューニングルーチンのパラメタ指定用入力ファイル

注：Xには対象となる AT 領域名が入る

4.2.1 入力ファイル

入力ファイルについては、ユーザ指定ファイルとシステム指定ファイルの両方があります。

ユーザ指定ファイル：

ABCLib_InstallParamDefX.dat : インストール時自動チューニングルーチンのパラメタ指定用

ABCLib_StaticParamDefX.dat : 実行起動前自動チューニングルーチンのパラメタ指定用

ABCLib_DynamicParamDefX.dat : 実行時自動チューニングルーチンのパラメタ指定用

システム指定ファイル：

インストール時ルーチン：なし

実行起動前ルーチン：

ABCLib_InstallParamX.dat

実行時ルーチン：

ABCLib_StaticParamX.dat 、 ABCLib_DynamicParamX.dat

4.2.2 出力ファイル

出力ファイルについては、システム指定ファイルしかありません。

システム指定ファイル：

インストール時ルーチン：

ABCLib_InstallParamX.dat

実行起動前ルーチン：

ABCLib_StaticParamX.dat

実行時ルーチン：

ABCLib_DynamicParamX.dat

4.2.3 入出力ファイルの形式

ユーザ指定ファイル、およびシステム指定ファイルの形式は、以下のようにします。

```
<形式> ::=
( <名前>
  ( <指定部> <設定部> )
  [ ( <指定部> <設定部> ) ]
  ...
)
[ <形式> ];
<指定部> ::= ( <形式> | パラメタ名 );
<設定部> ::= [ パラメタの値 ];
```

<名前> は、チューニング領域名、もしくは基本パラメタを指定します。基本パラメタを指定する場合は、BasicParam と書きます。

4.3 ユーザ指定ファイル上のパラメタとの衝突

ユーザ指定ファイルに指定されているパラメタを自動チューニングする場合（パラメタを決定しようとする場合）を、**パラメタ衝突**といいます。

パラメタ衝突する場合は、その自動チューニングを停止し、ユーザ指定のパラメタを強制的に設定します。

システムとしては、パラメタ衝突が起こる場合は、ユーザにとってデバックなどの理由から自動チューニング機能を停止したい状況であると想定します。逆にいうと、ユーザ指定ファイルにパラメタ情報を定義することでデバックが可能です。

4.4 記述式のネスティング

ここでは、指定子のネスティングについて定義します。

4.4.1 ネスティングの可能性と深さ

ネスティングができる自動チューニングの種類について、表 1 で定義します。

表 1 自動チューニングの種類によるネスティングの可能性

ネスティングする側 (上位部分)	ネスティングされる側 (下位部分)		
	install	static	dynamic
install		×	×
static			×
dynamic			

ネスティングができる機能の組み合わせを、表 2 で定義します。

表 2 機能によるネスティングの可能性

ネスティングする側 (上位部分)	ネスティングされる側 (下位部分)			
	define	variable	select	unroll
define				
variable				
select				
unroll	×	×	×	×

ネスティングの深さ (どれだけ入れ子にしているか) の上限は、現在のところ以下のようになっています。すなわち、3 つまでしか入れ子にはいけません。

ネスティングの深さ = 3 以下

4.4.2 パラメタ探索の順序 (拡張機能)

ネストされたパラメタの探索方法は、上位のチューニング領域で指定した方式で決まります。パラメタ探索の方式は、以下の注釈で指定できます。

補助指定子 <パラメタ探索法> ::= (全探索方式 A D - H O C 方式)
--

いま、チューニング領域に m 個のパラメタ P_i ($i=1,2,\dots,m$) があり、それぞれ N_i 個のパラメタを変動する必要があるとします。このとき、パラメタを以下のように表記します。

$$P = (V(P_1), V(P_2), \dots, V(P_m))$$

ここで、 $V(P_i)$ はパラメタ P_i の N_i 個あるパラメタのうちの 1 つを表します。

全探索方式 :

!ABCLib\$ search Brute-force

全探索方式では、全ての組み合わせについて調べます。すなわち、パラメタ P の組み合わせ全てについて探索する方式です。

したがって、パラメタの組み合わせの数は、 N_i となります。

A D - H O C 方式 :

!ABCLib\$ search AD-HOC

A D - H O C 方式では、パラメタ P の組み合わせの数全てについて探索しません。

あるパラメタに固定した初期値から探索を開始し、 P_m を変動させて最適なパラメタを見つけて固定し、つぎに P_{m-1} を変動させ最適なパラメタを見つけて固定し、というように処理を進めます。最終的に、 P_1 までこの処理を繰り返します。

したがって、パラメタの組み合わせ数は N_i となります。

【ネストされた指定子ごとに指定探索方式が異なる場合】

基本的には、下位の A T 領域から探索を開始して、上位の探索方式に合わせます。しかし、下位が A D - H O C 方式、上位が全探索方式の場合、A D - H O C 指定の A T 領域のパラメタは固定値として扱うものとします。

【プログラム例 8】

以下のネストされた処理で、どのようにパラメタの探索を行うか。

```
!ABCLib$ static variable (BL) region start  
!ABCLib$ name ABlockRoutine  
!ABCLib$ varied 1 from 16  
do iter=1, n, BL  
  
...  
!ABCLib$ static unroll (i,j) region start  
!ABCLib$ name Kernel1  
!ABCLib$ varied (i,j) from 1 to 32  
  do i=1+iter, n  
    do j=1+iter, n  
      do k=1+iter, n  
        .....  
      enddo  
    enddo  
  enddo  
!ABCLib$ static unroll (i,j) region end  
  
...  
!ABCLib$ static unroll (i,j) region start  
!ABCLib$ name Kernel2  
!ABCLib$ varied (l,m) from 1 to 32  
  do l=1+iter, n  
    do m=1+iter, n  
      do p=1+iter, n  
        .....  
      enddo  
    enddo  
  enddo  
!ABCLib$ static unroll (l,m) region end  
  
...  
enddo  
!ABCLib$ static variable (BL) region end
```

いま、パラメタの並びを (BL, (i, j), (l, m)) とします。

上記の例の場合では、チューニング領域 ABlockRoutine、Kernel1、Kernel2、全てにおいて全探索方式とします。このときのパラメタ探索は、

(1,(1,1),(1,1)), (1,(1,1),(1,2)),..., (1,(1,1),(1,32)),
(1,(1,1),(2,1)), (1,(1,1),(2,2)),..., (1,(1,1),(2,32)),
...,
(1,(1,2),(1,1)), (1,(1,2),(1,2)),..., (1,(1,2),(1,32)),

のように、 $16 \times 32 \times 32 \times 32 \times 32 = 1,677,216$ 通り探索することになります。

上記の例の場合で、チューニング領域 ABlockRoutine、Kernel1、Kernel2 で全て AD - HOC 方式とします。このときは、

(1,(1,1),(1,1)), (1,(1,1),(1,2)),..., (1,(1,1),(1,32)) : 最も高速なパラメタを決める (例: 8)
(1,(1,1),(1,8)), (1,(1,1),(2,8)),..., (1,(1,1),(32,8)) : 最も高速なパラメタを決める (例: 4)
(1,(1,1),(4,8)), (1,(1,2),(4,8)),..., (1,(1,32),(4,8)): 最も高速なパラメタを決める (例: 5)
(1,(1,5),(4,8)), (1,(2,5),(4,8)),...

のように探索をします。すなわち探索の回数は、 $16 + 32 + 32 + 32 + 32 = 144$ 通りのパラメタサーチをします。

上記の例の場合で、チューニング領域 ABlockRoutine は全探索方式、Kernel1、Kernel2 で AD HOC 方式とします。このときは、

(1,(1,1),(1,1)), (1,(1,1),(1,2)),..., (1,(1,1),(1,32)) : 最も高速なパラメタを決める (例: 8)
(1,(1,1),(1,8)), (1,(1,1),(2,8)),..., (1,(1,1),(32,8)) : 最も高速なパラメタを決める (例: 4)
(1,(1,1),(4,8)), (1,(1,2),(4,8)),..., (1,(1,32),(4,8)) : 最も高速なパラメタを決める (例: 5)
(1,(1,5),(4,8)), (1,(2,5),(4,8)),..., (1,(32,5),(4,8)) : 最も高速なパラメタを決める (例: 6)
(1,(6,5),(4,8)), (2,(6,5),(4,8)),...

のように探索をします。

すなわち探索の回数は、 $16 + 32 + 32 + 32 + 32 = 144$ 通りのパラメタサーチをします。

上記の例の場合で、チューニング領域 ABlockRoutine は AD-HOC 方式、Kernel1、Kernel2 で全探索方式とします。このときは、

(1,(1,1),(1,1)),(1,(1,1),(1,2)),..., (1,(1,1),(1,32)) ,
(1,(1,1),(2,1)),(1,(1,1),(2,2)),..., (1,(1,1),(2,32)) ,
...
(1,(1,1),(32,1)),(1,(1,1),(32,2)),..., (1,(1,1),(32,32)) : 最も高速なパラメタを決める (例(3,9))
(1,(1,1),(3,9)),(1,(1,2),(3,9)),..., (1,(1,32),(3,9)),
(1,(2,1),(3,9)),(1,(2,2),(3,9)),..., (1,(2,32),(3,9)),
...
(1,(32,1),(3,9)),(1,(32,2),(3,9)),..., (1,(32,32),(3,9)) : 最も高速なパラメタを決める (例(2,8))
(1,(2,8),(3,9)),(2,(2,8),(3,9)),..., (16,(2,8),(3,9)) : 最も高速なパラメタを決める (例 : 6)

のように探索をします。

すなわち探索の回数は、 $16+32*32+32*32 = 2,064$ 通りのパラメタサーチをします。

なお探索方式を指定しない場合 (デフォルト) では、以下のように設定されています。

機能名 : デフォルトの探索方式
define : (探索の必要なし)
variable : 全探索方式
select : AD - HOC方式
unroll : 全探索方式

5 おわりに

この利用の手引きでは、自動チューニングソフトウェア開発者用の自動チューニング処理記述ディレクティブ ABCLibScript の仕様と利用方法を説明しました。

ABCLibScript で記述された自動チューニング指示は、ソフトウェア開発者が有する特有の<知識>であるといえます。したがって、ソフトウェア技術者がもつ職人的技術知識の記述を ABCLibScript で行うことにより、その知識がソースコードを通じて他の技術者へ<継承>されるわけです。

この<知識>は、いままでは個別に獲得せざるを得ない事項であり、また、他人へ公開できなかった事項でした。このことにより、個人の所有するノウハウ・知識が、よくわからない職人的技術知識へとつながっていきました。

このような背景を考慮すると、ABCLibScript によるソースコード中への明示的な知識記述による波及効果は絶大であると著者は考えています。

参考文献

以下、関連論文を列挙します：

[A-1] Brewer, A.E., Portable High-Performance Supercomputing: High-Level Platform-Dependent Optimization, Ph.D Thesis, Massachusetts Institute of Technology (1994)

[A-2] Bilmes, J., Asanovic, K, Chin, C.-W. and Demmel, J.: Optimizing Matrix Multiply using PHiPAC: a Portable, High-Performance, ANSI C Coding Methodology, Proceedings of International Conference on Supercomputing 97, pp.340-347 (1997)

[A-3] 高田広章：特集「組み込みシステム開発の現状」、情報処理、Vol.38、No.10、pp.870—903 (1997)

[A-4] 黒田久泰、片桐孝洋、佃良生、金田康正：自動チューニング機能付き並列数値計算ライブラリ構築の試み 対称疎行列用の連立一次方程式ソルバを例にして、情報処理学会第57回全国大会講演論文集(1)、pp.1-10-1-11 (1998)

[A-5] Frigo, M.: A Fast Fourier Transform Compiler, Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation, Atlanta, Georgia, pp.169-180 (1999)

[A-6] Whaley, R., Petitet, A. and Dongarra, J.J.: Automated Empirical Optimizations of Software and the ATLAS project, Parallel Computing, Vol.27, pp. 3-35 (2001)

[A-7] 片桐孝洋、黒田久泰、大澤清、工藤誠、金田康正：自動チューニング機構が並列数値計算ライブラリに及ぼす効果、情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム、Vol.42、No.SIG 12 (HPS 4)、pp. 60-76 (2001)

[A-8] 直野健、山本有作：単一メモリ型インタフェースを有する自動チューニング並列ライブラリの構成方法、情報処理学会研究報告、No. 2001-HPC-87、pp.25-30 (2001)

[A-9] Ribler, R.L., Simitci, H. and Reed, D.A.: The Autopilot Performance-Directed Adaptive Control System, Future Generation Computer Systems, Special Issue (Performance Data Mining), Vol.18, No.1, pp. 175-187 (2001)

[A-10] Kuroda, H., Katagiri,T., and Kanada,Y: Knowledge Discovery in Auto-tuning Parallel Numerical Library, Progress in Discovery Science, Final Report of the Japanese Discovery Science Project. Lecture Notes in Computer Science 2281 Springer 2002, ISBN 3-540-43338-4, pp.628–639 (2002)

[A-11] Tapus, C., Chung, I.-H. and Hollingsworth, J. K.: Active Harmony : Towards Automated Performance Tuning, Proceedings of High Performance Networking and Computing (SC2002), Baltimore, USA (2002)

[A-12] 片桐孝洋、吉瀬謙二、本多弘樹、弓場敏嗣：実行起動前最適化層を有する自動チューニングソフトウェア構成方式の提案、2003年先進的計算基盤システムシンポジウム (Symposium on Advanced Computing Systems and Infrastructures (SACSIS)、SACSIS 2003 論文集、pp.159—160 (2003)

[A-13] Katagiri,T., Kise,K., Honda,H., and Yuba,T., : FIBER: A Framework of Installation, Before Execution-invocation, and Run-time Optimization Layers for Auto-tuning Software, 電気通信大学情報システム学研究科技術報告, UEC-IS-2003-3 (2003)

[A-14] 片桐孝洋、吉瀬謙二、本多弘樹、弓場敏嗣：FIBER：汎用的な自動チューニング機能の付加を支援するソフトウェア構成方式、第94回ハイパフォーマンスコンピューティング (HPC) 研究会、平成15年6月13日 (金) 情報処理学会研究報告 2003-HPC-94, pp. 1—6 (2003)

[A-15] Katagiri, T., Kise K., Honda, H., and Yuba,T.,: FIBER: A General Framework for Auto-Tuning Software, Springer LNCS 2858, pp.146–159, The Fifth International Symposium on High Performance Computing (ISHPC-V), Tokyo Fashion Town Building, Tokyo International Trade Center (Odaiba, Tokyo, JAPAN), October 20-22 (2003)

[A-16] 今村俊幸、直野健：性能安定化を目指した自動チューニング型固有値ソルバーについて、先進的計算基盤システムシンポジウムSACSIS 2003論文集、pp.145—152, (2003)

[A-17] 須田礼仁：ERXPP-数値ライブラリにより並列計算性能を簡易かつ適応的に引き出す方式の提案、情報処理学会研究報告、2003-HPC-96、pp.19—24 (2003)

[A-18] 高田広章：組み込みシステム開発の現状と課題、先進的計算基盤システムシンポジウム SACSIS 2003 チュートリアル資料 (2003)

[A-19] Katagiri,T., Kise,K., Honda,H., and Yuba,T.: Effect of Auto-tuning with User's Knowledge for Numerical Software, Proceedings of ACM Computing Frontiers (CF) 04, pp.12--25, Island of Ischia, Italy, 14 16 (2004)

[A-20] Cuenca, J., Gimenez, D., and Gonzalez, J.: Architecture of an Automatically Tuned Linear Algebra Library, Parallel Computing, Vol.30, pp.187—210 (2004)

[A-21] 今村俊幸、直野健：キャッシュ競合を制御する性能安定化機構内蔵型数値計算ライブラリについて、ハイパフォーマンスコンピューティングと計算科学シンポジウム HPC S 2004 論文集、pp.173—180 (2004)

[A-22] 直野健、今村俊幸、恵木正史：GRIDコンピューティング環境における行列ライブラリ向け性能保証方式の検討、ハイパフォーマンスコンピューティングと計算科学シンポジウム HPC S 2004 論文集、pp.51—58 (2004)

[A-23] 石井良規、片桐孝洋、本多弘樹、弓場敏嗣：Autopilot を用いた疎行列ソルバにおける実行時自動チューニング機構の設計、電子情報通信学会総合大会論文集、D-3-9、pp. 28、(2004)

[A-24] Eijkhout,V., Fuentes,E: Statistical Techniques for Algorithm Ranking In Self-Adapting Numerical Software, Eleventh SIAM Conference on Parallel Processing for Scientific Computing (PP04), Hyatt at Fisherman's Wharf, San Francisco, CA, USA, February 25, 2004, Organized Session of MS22, Multimethod Sparse Solvers for Large Scale Simulations (2004)

[A-25] 渡辺政彦：組み込みソフトウェア向け開発支援環境、情報処理、Vo.45、 No.1、pp.10—15 (2004)

[A-26] 片桐孝洋、吉瀬謙二、本多弘樹、弓場敏嗣：自動チューニング処理記述用ディレクティブ ABCLibScript の設計と実装、2004年先進的計算基盤システムシンポジウム (Symposium on Advanced Computing Systems and Infrastructures (SACSIS)、SACSIS 2004 論文集、pp.43—52 (2004)、および、自動チューニング処理記述用ディレ

クティブ ABCLibScript、電気通信大学情報システム学研究科技術報告、UEC-IS-2004-1 (2004) :

[A-27] 片桐孝洋、吉瀬謙二、本多弘樹、弓場敏嗣：自動チューニング処理記述用ディレクティブ ABCLibScript の設計と実装、2004年先進的計算基盤システムシンポジウム (Symposium on Advanced Computing Systems and Infrastructures (SACSIS)、SACSIS 2004 論文集、pp.43—52 (2004)

[A-28] 直野健、恵木正史：GRIDコンピューティング環境下での行列ライブラリにおける性能保証ライン算出の改良型アルゴリズム、2004年先進的計算基盤システムシンポジウム (Symposium on Advanced Computing Systems and Infrastructures (SACSIS)、SACSIS 2004 論文集、pp.295—304 (2004)

[A-29] 片桐孝洋、吉瀬謙二、本多弘樹、弓場敏嗣：ユーザ知識を活用するソフトウェア自動チューニングについて、第 回 ハイパフォーマンスコンピューティング (HPC) 研究会、平成 15 年 8 月 1 日 (金)、情報処理学会研究報告 2004-EVA-10、pp.19—24 (2004)

[A-30] マイクロソフト SQL Server 2000 :
<http://www.microsoft.com/japan/sql/default.msp>

[A-31] 東京大学 生産技術研究所 桑原研究室：交通流シミュレーション、ネットワークシミュレーションの容量パラメタの自動調整法：<http://www.transport.iis.u-tokyo.ac.jp/>

[A-32] 電子情報通信学会、ディペンダブルコンピューティング研究会、
<http://www.ieice.org/iss/dc/jpn/>

[A-33] 片桐孝洋、吉瀬謙二、本多弘樹、弓場敏嗣：データ再分散を行う並列 Gram-Schmidt 再直交化、情報処理学会論文誌：コンピューティングシステム、Vol.45、No. SIG 6 (ACS 6)、pp.75—85 (2004) :

[A-34] 直野健、猪貝光祥：マルチカラー逆反復法による直交化法とその性能、日本応用数理学会年会、オーガナイズドセッション：数値線形代数、中央大学後楽園キャンパス、2004年9月16日～18日

以下、関連プロジェクトを列挙します：

[B-1] PHiPAC プロジェクト； <http://www.icsi.berkeley.edu/~bilmes/hipac/>

[B-2] ATLAS プロジェクト； <http://www.netlib.org/atlas/index.html>.

[B-3] ILIB プロジェクト (HINTS プロジェクト)； <http://www.hints.org/>

[B-4] ABCLib プロジェクト； <http://www.abc-lib.org/>

[B-5] FIBER プロジェクト；

http://www.yuba.is.uec.ac.jp/~katagiri/FIBER_Prj/FIBER-Prj.html

[B-6] Autopilot プロジェクト：

<http://www-pablo.cs.uiuc.edu/Project/Autopilot/AutopilotOverview.htm>

[B-7] Active Harmony プロジェクト：<http://www.dyninst.org/harmony/>

[B-8] SANS プロジェクト； <http://icl.cs.utk.edu/sans/index.html>

[B-9] SALSA プロジェクト； <http://icl.cs.utk.edu/salsa/>

[B-10] BeBOP プロジェクト； <http://bebop.cs.berkeley.edu/>

[B-11] Sparsity プロジェクト； <http://www.cs.berkeley.edu/~yelick/sparsity/>

以下、F I B E Rに関連する特許出願を列挙します：

[C-1] 片桐孝洋：プログラム、記録媒体およびコンピュータ、日本国特許出願、
特願 2003-022792 (平成 15 年 1 月 30 日)

[C-2] 片桐孝洋：計算装置、計算方法、プログラムおよび記録媒体、日本国特許出願、
特願 2003-092592 (平成 15 年 3 月 28 日)。

[C-3] 片桐孝洋：計算装置、計算方法、プログラムおよび記録媒体、日本国特許出願、特願
2003-149701、平成 15 年 5 月 27 日、(特願 2003-92592 の国内優先
権出願)